ABSTRACT
The Linked Open Data (LOD) cloud is in danger of becoming a black box. Simple questions such as “What kind of datasets are in the LOD cloud?”, “In what way(s) are these datasets connected?” – albeit frequently asked – are at the moment still difficult to answer due to the lack of proper tooling support. The infrequent update of the static LOD cloud diagram adds to the current dilemma, since there is neither reliable nor timely-updated information to perform an interactive search, analysis or in particular visualization in order to gain insight into the current state of Linked Open Data. In this paper, we propose a new hybrid system which combines LOD Visualisation, Analytics and DiscovERy (LODVader) to aid in answering the above questions. LODVader is equipped with (1) a multi-layer LOD cloud visualization component comprising datasets, subsets and vocabularies, (2) dataset analysis components that extend the state of the art with new similarity measures and efficient link extracting techniques and (3) a fast search index that is an entry point for dataset discovery. At its core, LODVader employs a timely-updated index using a complex cluster of Bloom filters as a fast search index with low memory footprint. This BF cluster is able to efficiently perform analysis on link and dataset similarities based on stored predicate and object information, which – once inverted – can be employed to discover invalid links by displaying the Dark LOD Cloud. By combining all these features, we allow for an up-to-date, multi-dimensional LOD cloud analysis, which – to the best of our knowledge – was not possible before.

Keywords
Linked Open Data, Linksets, Bloom filter, RDF diagram
REST API, we created a front end using NodeJS\(^6\) which allows the visualization of the diagrams using the Data Driven Documents\(^6\) JavaScript library. One can try out LODVader following this link [http://lodvader.aksw.org\(^7\)]. The implementation of the REST API and the front end are open source and are available on GitHub.

The Manager is the central controller of the service and is responsible to serve the API calls and coordinate the processing components. Once a user submits a dataset for processing (using a description file), the Manager will fetch for URL of distributions (usually described by `dcat:downloadURL` or `void:dataDump`) and will dispatch the call to the RDF streaming module responsible for stream and parse the distributions. Distributions normally are dump files or SPARQL endpoint. The output of this phase is an array of triples \(<s, p, o>\) that is dispatched to the Index engine where the indexes are created. The Manager is also connected to a set of plugins. Plugins are modules with well defined functions that allow our framework to be extended for different use cases. As an example, the Analytics back end Plugin is responsible for determine the top N links between two datasets and the Search Plugin queries Bloom Filters (BF) in order to find subjects, predicates and objects. The plugins are all connected to the REST API, making possible an integration with a front end or even different frameworks.

A particular plugin that allows users to visualize the linked datasets is the Visualization plugin. This plugin provides JSON objects which characterize the current condition of the links between the distributions. Hence, using the front end provides a new visualization of the LOD-Cloud diagram (which can be seen in Figure 2) using multiple layers of data, i.e. distributions and subsets are within datasets and are represented by different edges of the graph. In addition to the classical view, where vertexes represents the amount of links between datasets, the LODVader framework allows different dataset visualizations which can help the user to perform analysis operations on the imported LOD cloud. For example it is possible to filter datasets based on their similarities (using Jaccard coefficient based on `rdf:type`, `owl:Classes` and general predicates), datasets and ontologies. Furthermore, subsets and distributions of a dataset are shown and grouped as clusters of bubbles.

The main difference between LODVader and lod-cloud\(^8\) is that instead of assuming that every distribution is in the same pay-level domain or sub-domain, LODVader precisely compare every object and subject for each source dataset and target dataset.

Moreover, we propose the novel concept of the Dark LOD diagram which can be seen in Figure 3. The Dark LOD diagram, visualizes links between objects of a source distribution which are not described as subjects in a target distribution.

Besides to be able to read and write RDF data (RDF is generated on the fly by Apache JENA), LODVader doesn't use a triplestore. Instead, LODVader uses a MongoDB as a database to store and fetch relevant data. MongoDB has shown fast and scalable enough to be used with BF as a central index for linked datasets.

3. LODVADER USAGE (DEMO PRESENTATION)

The following subsections describe the usage of the four current features of the LODVader Visualisation plugin and Manager. The presentation in the Demo Track will go through all of them showing the framework performance. For the first three examples, we demonstrate how to use the REST API (although all of them can be done using the font end), using the base URL "http://api.lodvader.aksw.org/", and in the fourth example we will demonstrate the usage of the LOD diagram.

3.1 Adding a dataset to the framework

In order to visualise a dataset via LODVader, the user can either create an entry at indexed repositories, such as [http://datahub.io](http://datahub.io) or [http://lov.okfn.org](http://lov.okfn.org) or she should [http://lod-cloud.net/](http://lod-cloud.net/)
provide a description file which contains metadata describing
the datasets to be streamed as shown in Figure 4. LODVader is capable of consuming RDF descriptions of datasets
according to DCAT [13], VoID [14] and DataID [2], fetching for
URL described by dcat:downloadURL or void:dataDump.

The request parameters to the API are two: descriptionFileURL which should contain the URL of the description
file and format which contains the serialization format
(available options are: ttl, nt, rdfxml or jsonld) of the
description file. The REST API returns an array of JSON
objects which contains a list of fetched distributions. Each
JSON object describes the status of the streaming process,
error messages and amount of triples read.

3.2 Finding data within a dataset

LODVader uses Boom filters to store indexes of
subject, predicate and object. Thus, it’s possible to query whether a
dataset or a vocabulary contains a particular resource. The
parameters used are:

- searchSubject: Parameter used to find a particular
  subject resource in a dataset
- searchProperty: Parameter used to find a particular
  property resource in a dataset
- searchObject: Parameter used to find a particular
  object in a dataset. Note that literals are not allowed here.
- searchVocabulary: Boolean value that defines whether
to filter only vocabularies or datasets. Omitting it will
  search both.

As an example, the URL [14] would return an array of JSON
objects of all datasets which contains the DBpedia subject
Hawaii.

3.3 Retrieving VoID:linkset

We will also show in the presentation that LODVader also
allows to retrieve RDF data in the void:linkset format.
Essentially two parameters are used: source and target
dataset. Both parameters should have a value if the user
wants to compare two specific datasets, otherwise only the
source is mandatory. The described RDF includes the URI of
the source dataset, target dataset, number of triples and the
provenance using the using the Prov-O ontology[17]. The
REST response should be a RDF represented using turtle
format as shown in Figure 5.

http://www.w3.org/TR/vocab-dcat/
http://www.w3.org/TR/void/
http://www.w3.org/TR/prov-o/

3.4 Customized LOD Cloud graph

Next, we are going to show the visualization module (in-
cluded in the front end GUI) and capability of browsing
and creating customized LOD Cloud. The API allows to
retrieve a JSON object where links between datasets are
represented by edges and datasets are represented by ver-
texes. This structure is useful when a customized diagram
should be rendered. The parameters used are:

- dataset: The array of datasets that LODVader should
crawl. The returned JSON object will contain the rep-
  resentation of the datasets consisting of the array plus
  the datasets which contains indegree and outdegree
  links.
- linkType: Describes the type of the retrieved links.
  Should be showLinks, showSimilarities or show-
  DarkLOD. Case the similarity option is chosen, two
  parameters (from and to) are optional and represent
  the similarity range (from 0 to 1).

Along with demonstrating the usage of the parameters,
we will show the creation of a graph similar to the Figure 2
and demonstrate the usage of filters (e.g. how to filter vo-
cabularies from the diagram) and the behavior of the graph
when a dataset contains multiple subsets or distributions.
The last feature to be presented in the visualization mod-
ule is the Dark LOD Diagram and its features. We will
show and explain how we create a cloud which contains only
datasets with invalid links to debug the LOD Cloud. Fig-
ure 6 shows the above-mentioned options in the LODVader
interface. Note that we also index vocabularies and treat
a triple with rdf:type as a link between a dataset and its
schema in LODVader.

4. EVALUATION

In order to evaluate the performance of the LODVader,
we first compared indexing data using BF with HashMap
Search (HS) and Binary Search Tree (BST) w.r.t. memory
usage for each structure to index distributions. Secondly, we
compared LODVader with OpenLink Virtuoso [18] w.r.t. time
to load and index triples, time to make searches, amount
of data on the storage. We are aware that LODVader is
not a triplestore, and do not store sufficient data do make
a complete SPARQL query. However, making complexes

http://virtuoso.openlinksw.com/
store the same triple twice, considering that it’s a graph structure. The problem is emphasized since each framework deals differently with erroneous data. For example, a bad RDF structure might be accepted by a particular framework, but completely ignored by others. Hence, when we deal with large amount of triples, it’s difficult to have the exact number of resources in different frameworks.

Table [1] shows the results w.r.t for loading and indexing triples and the total space used in the hard drive. Open-Link Virtuoso loaded triples in 24:02:23, while LODVader 06:32:01, making the execution 3.67 times faster. The amount of data stored was 84,28Gb for OpenLink Virtuoso and 4,03Gb for LODVader.

5. ACKNOWLEDGEMENTS

This paper’s research activities were funded by grants from the FP7 & H2020 EU projects ALIGNED (GA-644055), LIDER (GA-610782), FREME (GA-644771), Smart Data Web (GA-01MD15010B) and CAPES foundation (Ministry of Education of Brazil) for the given scholarship (13204/13-0).

6. REFERENCES

